Ensemble Techniques

Homework 3

Amber Barksdale

Machine Learning
CS-GY 6923
NYU Tandon School of Engineering
April 2022

Contents

1 Introduction to Data Set

The data set used in this analysis is called "Dry Bean Dataset", which can be found in the University of California Irvine Machine Learning Repository. The data set comes from a research paper titled Multiclass Classification of Dry Beans Using Computer Vision and Machine Learning Techniques (Koklu, M. and Ozkan, I.A., 2020)[2]. More about this dataset, the significance of dry beans in the Turkish economy, and the necessity for building an accurate bean classifier can be found in the paper itself, or the first phase of this project titled "Exploratory Data Analysis".

1.1 Third Phase

The purpose of this phase of the Dry Beans data analysis project is to perform three ensemble techniques, determine how each effect variance and bias, and compare them. The list of techniques to choose from include:

- Cross-Validation
- Leave-One-Out Cross-Validation
- Bagging
- Random Forest
- Boosting
- Stacking

2 Cross Validation

Cross-Validation can be used to determine the performance of a given model on unseen, or test, data. In this iteration, the data is divided into groups called *folds*. The model is trained on every grouping of folds and tested on the one that remains. The performance - in this case, Root Mean Squared Error (RMSE) - is recorded and then averaged over the number of folds. Multiple models or formulas are run, and the average performance of the cross-validated models can be compared. The "best" model would be the one with the lowest RMSE value. This model may be the best at predicting based on data it has not encountered before, which is the goal [3].

2.1 Comparison to Previous Project

In the second phase of the project, titled "Individual Classifiers", each model (as long as it required) was split into train and test sets. However, cross-validation goes an extra step and splits these train and test sets into smaller train and test sets, and the model learns over multiple combinations of these smaller sets.

2.2 Method

A large amount of this code was modified from this tutorial on cross-validation.

2.2.1 Split Data

First, the data must be split into the training and test sets. This is the same process as from the previous phase of the project. In this case, we are doing an 20/80 split. Then, we create the folds for cross-validation.

```
# CROSS VALIDATION
library(groupdata2)
library(checkmate)
library(knitr)
library(naivebayes)
library(psych)
library(hydroGOF)

# Make this reproducible
set.seed(4321)

# Split data in 20/80 (percentage)
parts <- partition(drybeans, p=0.2, cat_col="Class")

test_set <- parts[[1]]
train_set <- parts[[2]]

# Create folds for cross-validation
train_set <- fold(train_set, k=4, cat_col="Class")
```

2.2.2 Initialize Formulas

For simplicity, we will create some variables that hold different formulas. Adding in the formula can be done by hand.

These formulas were based off the "Variable Importance" from Bagging, as well as the PCA Rotation from the first phase of the project. Formulas **m0** and **m3** are based off the three and five most important variables according to "Variable Importance" (see Figure 1), respectively. Formulas **m1** and **m2** are based off the first three and five most important variables according to the PCA Rotation. Formula **m4** is based off the three least important variables - which happen to be the same from both the PCA Rotation and the Variable Importance graph, just to see the level of error when using factors that are not as good of predictors.

```
m0 <- 'Class~Compactness+ShapeFactor1+AspectRation'
m1 <- 'Class~MajorAxisLength+ShapeFactor2+Perimeter'
m2 <- 'Class~MajorAxisLength+ShapeFactor2+Perimeter+
```

```
        EquivDiameter+ConvexArea+Area'
m3 <- 'Class~Compactness+ShapeFactor1+AspectRation+
        ShapeFactor3+MajorAxisLength+Area'
m4 <- 'Class~Extent+Solidity+ShapeFactor4'
```

2.2.3 Cross-Validation Function

This function is simply a more automated way to run the k-folds on the different models. Note that the `naive_bayes` function has a default call and a "class: formula" call. I am using the "class: formula" call here, and it may throw errors or cause problems if the `naive_bayes` function is not called individually with the specific formula input. In other words, trying to run `naive_bayes` with a variable standing in for the formula, as it is done here, does not seem to evoke the "class: formula" call, and rather the default call.

```
crossvalidate <- function(data, k, formula){
  # 'data' is the training set with the ".folds" column
  # 'k' is the number of folds we have
  # 'formula' is a string describing a formula

  # Initialize empty list for recording performances
  performances <- c()

  # One iteration per fold
  for (fold in 1:k){

    # Create training set for this iteration
    # Subset all the datapoints where .folds doesn't match current fold
    training_set <- data[data$.folds != fold,]

    # Create test set for this iteration
    # Subset all the datapoints where .folds matches current fold
    testing_set <- data[data$.folds == fold,]

    # Train model on training set
    model <- naive_bayes(formula, training_set)

    # Test model
    predicted <- predict(model, testing_set, allow.new.levels = TRUE)

    # Root Mean Square Error between the predicted and the observed
    RMSE <- rmse(
        as.numeric(
            factor(predicted)
        ),
        as.numeric(
            factor(
```

```
                testing_set[['Class']]
            )
        )
    )
    # Add the RMSE to the performance list
    performances[fold] <- RMSE
  }
  # Return the mean of the recorded RMSEs
  return(c('RMSE' = mean(performances)))
}
```

2.2.4 Run on all formulas

```
"Class ~ Compactness + ShapeFactor1 + AspectRation"
crossvalidate(train_set, k = 4, formula = m0)
RMSE = 0.9916439

"Class ~ MajorAxisLength + ShapeFactor2 + Perimeter"
crossvalidate(train_set, k = 4, formula = m1)
RMSE = 1.077764

"Class ~ MajorAxisLength + ShapeFactor2 + Perimeter +
         EquivDiameter + ConvexArea + Area"
crossvalidate(train_set, k = 4, formula = m2)
RMSE = 1.064235

"Class ~ Compactness + ShapeFactor1 + AspectRation +
         ShapeFactor3 + MajorAxisLength + Area"
crossvalidate(train_set, k = 4, formula = m3)
RMSE = 0.9806129

"Class ~ Extent + Solidity + ShapeFactor4"
crossvalidate(train_set, k = 4, formula = m4)
RMSE = 2.112544
```

As we can see, the last formula **m4**, which was based on the three least important predictors, has the highest RMSE. On the other hand, it looks like the "best" model is **m3**, so we will run this on the entire training and test set to check for overfitting.

2.2.5 Check for Overfitting

```
model <- naive_bayes(
         Class ~ Compactness + ShapeFactor1 + AspectRation +
         ShapeFactor3 + MajorAxisLength + Area,
         train_set)
```

```
prediction <- predict(model, test_set, allow.new.levels=TRUE)

RMSE <- rmse(
        as.numeric(
            factor(prediction)
        ),
        as.numeric(
            factor(
                test_set[['Class']]
            )
        )
    )

RMSE = 1.014237
```

The RMSE while running the "best" model, **m3**, is 1.014237, which is pretty close to what it was during cross validation (0.9806129). This means that we don't have to worry about this model overfitting.

2.3 Effect of Cross Validation on Bias & Variance

In general, as the number of folds (k) increases, the variance decreases [4]. Oftentimes, k is set to 5 or 10, "as these values have been shown empirically to yield test error rate estimates that suffer neither from excessively high bias nor from very high variance" [1].

3 Bagging

Bootstrap Aggregating, or Bagging, is an ensemble technique that helps reduce variance and avoid overfitting [6]. The general description of this technique is to create m new data sets, each of size $n*$. These data sets are generated by sampling from the original data set uniformly with replacement. Due to sampling with replacement, there may be some repeated observations in the new data set(s). Sampling with replacement allows each "bootstrap" data set is independent, as there is no dependency on previous data sets that have been created [6]. After m new data sets have been created, m models are fitted using those new sets and combined.

3.1 Method

Much of the code used for this Bagging method was based off and modified from this tutorial on Bagging in R.

```
library(dplyr)       # Data manipulation
library(e1071)       # Calculating variable importance
library(caret)       # General model fitting
```

```
library(rpart)        # Fitting decision trees
library(ipred)        # Fitting bagged decision trees

# Allows us to reproduce this example
set.seed(4321)

# Fit the bagged model
bag <- bagging(
  formula = as.factor(Class) ~ .,
  data = drybeans,
  nbagg = 150,
  coob = TRUE,
  control = rpart.control(minsplit = 2, cp = 0)
)

# Display the model
bag

Out-of-bag estimate of misclassification error:  0.0755
```

Note that the two arguments `minsplit` and `cp` allow the individual trees to grow deeply. This leads to high variance and low bias. After applying bagging, we can reduce the variance of the final model while keeping the bias low [8].

From the output of the model we can see that the out-of-bag estimated RMSE is 0.0755. This is the average difference between the predicted value for Class and the actual observed value.

3.1.1 Calculate Variable Importance

We can calculate the total reduction in Residual Sum of Squares (RSS) averaged over all of the trees [8]. This will allow us to visualize the importance of each predictor variable. Note that the larger the value, the more important the predictor [8].

```
# Calculate variable importance
VI <- data.frame(var=names(drybeans[,-17]), imp=varImp(bag))

# Sort in descending order
VI_plot <- VI[order(VI$Overall, decreasing=TRUE),]

# Plot
barplot(VI_plot$Overall,
        names.arg=rownames(VI_plot),
        horiz=TRUE, col='steelblue',
        xlab='Variable Importance',
        las=2, cex.names=0.5)
```

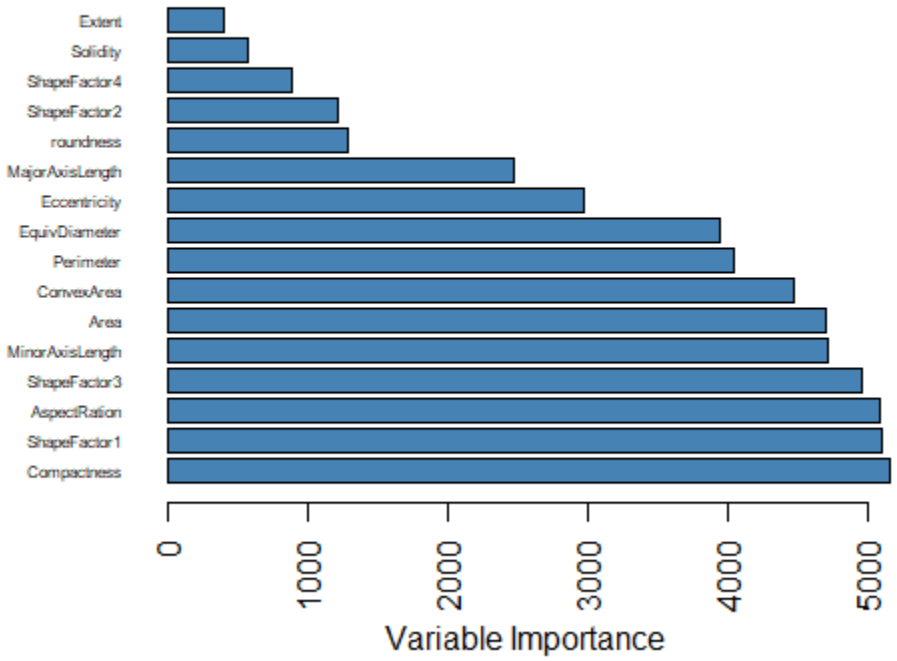

Figure 1: The figure shows all 16 variables in order of importance to the predictor. This graph suggests that Compactness, Shape Factor 1, and Aspect Ratio are the top three most important predictors.

3.2 Comparison to Previous Project

In the previous phase of this project, a single decision tree model was built and trained, and no bagging was performed. The decision tree was certainly one of the more poorly-performing models from the variety that were trained. Bagging is a method of improving the decision tree.

3.3 Effect of Bagging on Bias & Variance?

Bagging reduces variance and helps prevent overfitting, which is useful in high variance classifiers such as Decision Trees [5].

4 Random Forest

The goal of the Random Forest method is to correct the tendency of a Decision Tree to overfit the training set[7]. In a classification task such as the one with this data set, the output of the forest is the class that is reported by the majority of trees. The training method for a random forest actually applies the general ideas of Bagging [7].

4.1 Method

Much of the code in this section was modified or influenced by this tutorial on Random Forests in R.

```
library(randomForest)
library(groupdata2)
library(caret)

# Ensure this is reproducible
set.seed(4321)

# Split data into TEST and TRAIN
index <- createDataPartition(y=drybeans$Class,p = 0.7,list= ALSE)
train <- drybeans[index,]
test <- drybeans[-index,]

# Run Random Forest
rf <- randomForest(x=train, y=as.factor(train$Class), proximity=TRUE)

Call:
 randomForest(x = train, y = as.factor(train$Class), proximity=TRUE)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 4

```

```
        OOB estimate of  error rate: 0.05%
Confusion matrix:
          BARBUNYA BOMBAY CALI DERMASON HOROZ SEKER SIRA  class.error
BARBUNYA       924       0    0        2     0     0    0 0.0021598272
BOMBAY           0     366    0        0     0     0    0 0.0000000000
CALI             0       0 1139        0     0     2    0 0.0017528484
DERMASON         0       0    0     2483     0     0    0 0.0000000000
HOROZ            0       0    0        0  1349     1    0 0.0007407407
SEKER            0       0    0        0     0  1419    0 0.0000000000
SIRA             0       0    0        0     0     0 1846 0.0000000000
```

Out of bag error is 0.05%, so the train data set model accuracy is suggested to be around 99%.

The `rf.ntree` call reports that 500 trees were grown.

4.2 Comparison to Previous Project

Using a Random Forest model was a considerable improvement over the previous single Decision Tree, which had a reported accuracy of approximately 84%. The reported accuracy for this Random Forest is approximately 90%, which is a great improvement.

```
# Predict on Test data
pre <- predict(rf, test)

# Confusion Matrix & Statistics
confusionMatrix(pre, as.factor(test$Class))

          Reference
Prediction BARBUNYA BOMBAY CALI DERMASON HOROZ SEKER SIRA
  BARBUNYA      393       2    0        0     0     0    0
  BOMBAY          0     154    0        0     0     0    0
  CALI            2       0  489        0     0     0    0
  DERMASON        1       0    0     1063     2     0    0
  HOROZ           0       0    0        0   575     1    0
  SEKER           0       0    0        0     1   607    0
  SIRA            0       0    0        0     0     0  790

Overall Statistics

               Accuracy : 0.9978
                 95% CI : (0.9958, 0.999)
    No Information Rate : 0.2605
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.9973

```

```
26  Mcnemar's Test P-Value : NA
```

4.3 Effect of Random Forest on Bias & Variance

The goal of a random forest is to reduce the overall variance. The random forest averages many deep Decision Trees, where the trees themselves have low bias but high variance. The predictions of a single tree may be sensitive to noise, but it is argued that the average of many trees does not exhibit this [7]. Thus, averaging over the trees helps decrease the variance (at a cost of a small increase in bias)[7]. Overall, the Random Forest technique has been shown to greatly improve the performance of a model.

5 Comparing the Three Methods

Although all the ensemble techniques were developed with the goal of decreasing bias and variance, of course each has pros and cons.

5.1 Cross Validation

The main issue with Cross Validation is that the k value needs to be determined based on the data set and other context. Although $k = 5$ - 10 has been suggested as a good parameter, there is no fixed k that will work for all data sets and models.

5.2 Bagging

Bagging has been vaguely described as "preventing overfitting", but from the resources I was able to find, there was little explicit definition as to what degree Bagging reduces variance and bias. Intuitively, though, this technique makes sense to improve models that typically suffer from high variance (such as Decision Trees).

5.3 Random Forest

Another ensemble technique that is often paired with Decision Trees, Random Forest is also shown (albeit vaguely) to improve performance and decrease variance at the expense of a small increase in bias. Random Forest, like many other ensemble techniques, can also be applied to other models such as Multinomial Logistic Regression and Naïve Bayes classifiers.

5.4 Overall

Overall, it is clear to see that all three Ensemble Techniques are ways to improve the basic model (whichever one is being used). The idea of an ensemble technique naturally leads to ideas of even more complicated models such as neural networks and deep learning to see even more improvement in accuracy of prediction. However, this all comes at some computational cost.

5.5 Final Notes

The full R script, including the code for Exploratory Data Analysis and Individual Classifiers (the first two phases of this project), can be found in the remote IBM machine or at my GitHub repository.

6 R Script

```
# PART 3: ENSEMBLE TECHNIQUES

# BAGGING
library(dplyr)       #for data wrangling
library(e1071)       #for calculating variable importance
library(caret)       #for general model fitting
library(rpart)       #for fitting decision trees
library(ipred)       #for fitting bagged decision trees

#make this example reproducible
set.seed(4321)

#fit the bagged model
bag <- bagging(
  formula = as.factor(Class) ~ .,
  data = drybeans,
  nbagg = 150,
  coob = TRUE,
  control = rpart.control(minsplit = 2, cp = 0)
)

#display fitted bagged model
bag

#calculate variable importance
VI <- data.frame(var=names(drybeans[,-17]), imp=varImp(bag))

#sort variable importance descending
VI_plot <- VI[order(VI$Overall, decreasing=TRUE),]

#visualize variable importance with horizontal bar plot
barplot(VI_plot$Overall,
        names.arg=rownames(VI_plot),
        horiz=TRUE,
        col='steelblue',
        xlab='Variable Importance',
        las=2,
```

```
        cex.names=0.5)

library(Metrics)
actual <- drybeans$Class
predicted <- predict(bag)
act <- as.vector(as.factor(actual))
pred <- as.vector(as.factor(predicted))
a <- factor(act)
p <- factor(pred)
a <- as.numeric(a)
p <- as.numeric(p)

bias(a,p) # 0.00286

# CROSS VALIDATION
library(groupdata2)
library(checkmate)
library(knitr)
library(naivebayes)
library(psych)
library(hydroGOF)

#make this example reproducible
set.seed(4321)

# Split data in 20/80 (percentage)
parts <- partition(drybeans, p=0.2, cat_col="Class")

test_set <- parts[[1]]
train_set <- parts[[2]]

# Create folds for cross-validation
train_set <- fold(train_set, k=4, cat_col="Class")

# Order by .folds
train_set <- train_set %>% arrange(.folds)

# Create possible formulas
m0 <- 'Class ~ Compactness + ShapeFactor1 + AspectRation'
m1 <- 'Class ~ MajorAxisLength + ShapeFactor2 + Perimeter'
m2 <- 'Class ~ MajorAxisLength + ShapeFactor2 + Perimeter +
       EquivDiameter + ConvexArea + Area'
m3 <- 'Class ~ Compactness + ShapeFactor1 + AspectRation +
       ShapeFactor3 + MajorAxisLength + Area'
m4 <- 'Class ~ Extent + Solidity + ShapeFactor4'


```

```r
# Cross-Validate
crossvalidate <- function(data, k, formula, dependent){
  # 'data' is the training set with the ".folds" column
  # 'k' is the number of folds we have
  # 'formula' is a string describing a formula
  # 'dependent' is a string with the name of the
  #  score column we want to predict

  print("Formula is: ")
  print(formula)

  # Initialize empty list for recording performances
  performances <- c()

  # One iteration per fold
  for (fold in 1:k){

    # Create training set for this iteration
    # Subset all the datapoints where .folds does not match the current fold
    training_set <- data[data$.folds != fold,]

    # Create test set for this iteration
    # Subset all the datapoints where .folds matches the current fold
    testing_set <- data[data$.folds == fold,]

    # Train model on training set
    model <- naive_bayes(
             Class ~ Extent + Solidity + ShapeFactor4,
             training_set)

    ## Test model

    # Predict the dependent variable in the testing_set with the trained model
    predicted <- predict(model, testing_set, allow.new.levels = TRUE)


    # Get the Root Mean Square Error between the predicted and the observed
    RMSE <- rmse(as.numeric(factor(predicted)),
                 as.numeric(factor(testing_set[['Class']])))

    # Add the RMSE to the performance list
    performances[fold] <- RMSE

  }

  # Return the mean of the recorded RMSEs
  return(c('RMSE' = mean(performances)))
```

```
132
133   # return(performances)
134
135 }
136
137 crossvalidate(train_set, k = 4, formula = m0, dependent = 'Class')
138 crossvalidate(train_set, k = 4, formula = m1, dependent = 'Class')
139 crossvalidate(train_set, k = 4, formula = m2, dependent = 'Class')
140 crossvalidate(train_set, k = 4, formula = m3, dependent = 'Class')
141 crossvalidate(train_set, k = 4, formula = m4, dependent = 'Class')
142
143 # RANDOM FOREST
144 library(randomForest)
145 library(groupdata2)
146 library(caret)
147
148 #make this example reproducible
149 set.seed(4321)
150
151 # Split data into TEST and TRAIN
152 index <- createDataPartition(y = drybeans$Class,p = 0.7, list = FALSE)
153 train <- drybeans[index,]
154 test <- drybeans[-index,]
155
156 # Run Random Forest
157 rf <- randomForest(x=train, y=as.factor(train$Class), proximity=TRUE)
158
159 # Predict on Test data
160 pre <- predict(rf, test)
161
162 # Confusion Matrix & Statistics
163 confusionMatrix(pre, as.factor(test$Class))
```